基于 Spark 的两表等值连接过程优化 *

张子栋 [1]，郑延斌 [2]

(1. 集美大学 计算机工程学院, 福建 厦门 361021; 2. 河南师范大学 计算机与信息工程学院, 河南 新乡 453007)

摘　要：在数据统计分析查询中表间的等值连接是常用的操作之一，但代价较高。大数据环境下大表之间等值连接的效率更低。为了解决该问题，提出了一种基于 Spark 的两表等值连接过程优化方法。首先根据数据价值密度特征构建 Bloom Filter 完成表的过滤操作；其次结合 Simi-Join 和 Partition Join 两者的优势，对过滤后的单侧表使用贪心算法进行拆分；最后对拆分后的子集进行连接，因此把两大表的连接过程转换为分阶段进行的两小表连接。代价分析和实验结果表明该算法与现有基于 Spark 的连接操作相比不仅在性能上得到了提升而且当出现数据倾斜时对算法效率影响较小。

关键词：Spark；等值链接；大数据；优化；拆分

中图分类号：TP391　　doi: 10.3969/j.issn.1001-3695.2017.08.0710

Optimization of two-table equivalent connection process based on Spark

Zhang Zidong[1], Zhang Yanbin[2]

(1. *College of Computer Engineering*, *Jimei University*, *Xiamen Fujian* 361021, *China*; 2. *College of Computer & Information Technology Henan Normal University*, *Xinxiang Henan* 453007, *China*)

Abstract: The equivalence connection between tables in the statistical analysis of data is one of the commonly used operations, but the price is relatively high. In big data environment, the connection of large scale data tables is less efficient. In order to solve this problem, this paper proposed a method for optimization of two-table equivalent connection process based on Spark: first, constructed the Bloom Filter to complete the filtering operation according to the low density of data density; secondly combined the advantages of Simi-Join method and Partition Join methods, the he greedy algorithm Splitting methods is used for the filtered unilateral table; lastly joined the split subsets . Then the connection process of two big tables was changed into two stages of the two small table connection, Cost analysis and experiments show that the proposed algorithm has improved performance compared with the existing Spark-based connection operation performance and data tilt.

Key Words: Spark; equivalent connection; large data analysis; optimize; split

0　引言

在大数据背景下，涌现出了 Hadoop、Spark 等计算框架。作为一种分布式内存并行计算框架，Spark[1]以弹性分布式数据集（resilient distributed datasets,RDD）[2]为数据结构，并支持迭代计算，在大数据处理中表现出较好的性能。Spark 环境中经常会执行数据统计分析、查询等任务，其中等值连接是常用的但是代价较高的操作之一。尤其是大数据环境下数据表规模巨大，大表之间的等值关联操作效率更加低下。

Spark 中对表等值连接的操作是 RDD 的执行操作[2]，主要采用 Spark SQL[3]组件中的 Broadcast Join、Shuffle Hash Join、Sort Merge Join 等方式。Broadcast Join 方式是将其中一张数据表通过广播变量的方式广播到 Spark 所有集群节点上，然后再执行 Join 操作。Broadcast Join 方式由于每个 Executor 都会接受到表的全部数据，占用一部分内存空间来储存接受的数据，以存储为代价避免了 shuffle 操作。但是该方式局限性明显，只适合大表对小表或者小表间的 Join。其他的连接方式如 Shuffle Hash Join 和 Sort Merge Join 等，都会对连接属性 Key 进行重划分[4]，会涉及 Shuffle。如果数据量大时，Shuffle 会带来大量的网络通信和磁盘 I/O，并且数据分布难以预知。当数据节点上数据倾斜问题出现时，会造成局部节点作业时间长、计算量较大，因而导致整体作业时间长、局部节点出现 OOM 和计算资源浪费等问题。由大数据的数据特征知道，数据价值密度低，进行连接操作的两表通常存在大量不需要 join 的数据元组。

Spark/MapReduce 编程中常用的连接算法有广播连接(map side join)和重划分连接(reduce side join)两种，但是它们都有局

基金项目：河南省科技攻关项目（132102210537，132102210538）；河南省软科学项目（142400411001）

作者简介：张子栋（1980-），男，河南新乡人，讲师，硕士，主要研究方向为多智能体系统；郑延斌（1964-），男（通信作者），河南内乡人，教授，博士，主要研究方向为虚拟现实、多智能体系统（zybcgf@163.com）.

限性。Xu 等[5]基于 MapReduce 模型提出了 Semi Join，其基本思路是选取一个小表 R，将其参与 join 的 key 抽取出来，保存到文件 F 中（F 文件很小可以放到内存中）。在 map 阶段，使用 DistributedCache 将 F 复制到各个 TaskTracker 上，然后将另一个表 Q 中不在 F 中的 key 对应的记录过滤掉，剩下的 reduce 阶段的工作与 reduce side join 相同；Ramesh 等[6]提出了一种基于 Bloom Filter 的等值连接算法，该算法采用 Bloom Filter 对待连接的数据集进行过滤操作，减少 Shuffle 阶段的数据量。然而其中的等值连接方法是基于 MapReduce 计算框架，将处理的中间结果存储在磁盘中，与 Spark 的机制不同，因此整体处理的效率较慢，而且其没有考虑数据倾斜；Zhou 等人[7]指出数据倾斜是因为 Hadoop 本身无法感知 mapper 端输出数据的分布情况，导致 reducer 的负载不均衡，影响连接执行的效率；Gufler 等人[8,9]提出了两种根据采样结果来确定划分函数的方法，从而保证 reducer 负载均衡。如果能够对 mapper 任务输出的中间结果进行统计分析，确定数据分布情况，从而设计适应的分区函数和数据分发机制，就能确保每个 reducer 的负载均衡；卞昊穹等人[10]提出了一种基于 Spark 的等值连接的优化方法，其优化思想是：首先获取 Fact 表连接属性的无重集合 FactUK，并记录对应记录的位置:其次用 FactUK 与 Dim 表进行连接，生成 JoinedUK。最后根据 JoinedUK 中记录的位置将 JoinedUK 与 Fact 表进行组装，得到最终结果。但是该方法只适用于大表与小表之间等值连接的情况，且默认数据表均可以缓存到内存中，然而在实际应用中，大部分数据表是很难完全缓存到内存中。

为了解决以上连接过程中的问题，本文提出了一种基于 Spark 的两表等值操作优化算法。该算法首先对数据表构建标准 BloomFilter[11]进行过滤；其次结合 Simi-Join 和 Partition Join 两者优势，通过对一侧表连接属性 Key 的分区信息和另一侧表 Key 数据倾斜[12]情况的分析结果，做为贪心策略的启发选择；最后对过滤后的单侧表使用贪心算法[13]]进行拆分，并对拆分后的子集进行连接。从而把两大表的连接过程转换为分阶段进行的两小表连接，并得到连接结果。代价分析和实验表明了本文的算法比基于 Spark 的连接操作在性能上得到了提升，当出现数据倾斜时对算法的影响较小。

1　基于 Spark 的两表等值连接优化方法

在对两大表等值连接过程中，首先要对两表中存在的大量不符合连接条件的元组进行过滤优化，降低整体数据量。针对 Spark 中 Broadcast Join、Shuffle Hash Join、Simi-Join 等的局限性，提出了一种在 Spark 上做等值连接的优化方法，分为两个阶段如图 1、2 所示，图中的符号如表 1 所示。

1.1　过滤连接表

因为 Bloom Filter 具有占用时空间小，运算快速快等优点，该阶段采用标准的 Bloom Filter 进行数据过滤。

抽取两个数据表（RDD_a、RDD_b）的连接属性 join Key，得到 Joinkey_a 和 Joinkey_b 两个新的 RDD，然后使用 Spark 自带的 distinct 操作对这两个 RDD 进行去重，得到两个无重的数据集 Joinkey_a 和 Joinkey_b。使用 Spark 类库的 BloomFilter 分别对无重集 JoinKey_a 和 JoinKey_b 的进行压缩处理，并计算得到两个位数组 BF_A 和 BF_B，再对 BF_A 和 BF_B 进行 and 运算，生成最终的过滤位数组 BF。使用 BF 分别对需要连接的两张数据表进行过滤操作，过滤掉不满足 join 连接的数据记录，生成两个数据表 RDD_a_f 和 RDD_b_f。

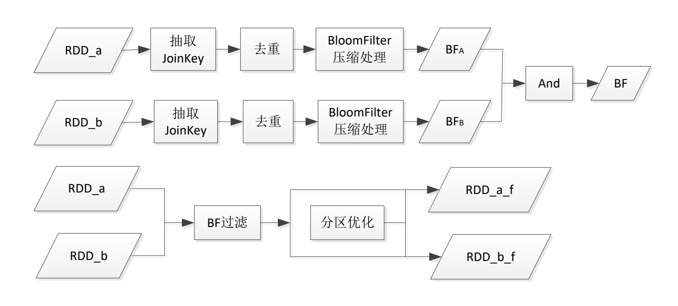


图 1　算法第一阶段：过滤连接表

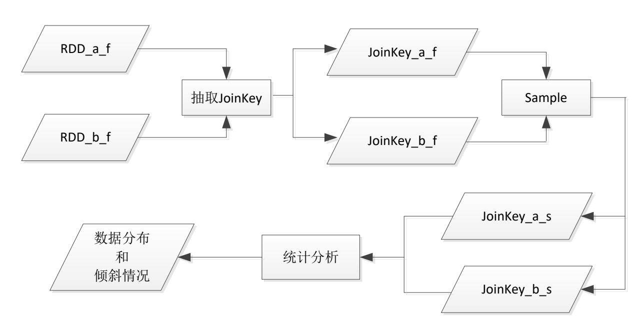


图 2　算法第二阶段：采样统计数据分布

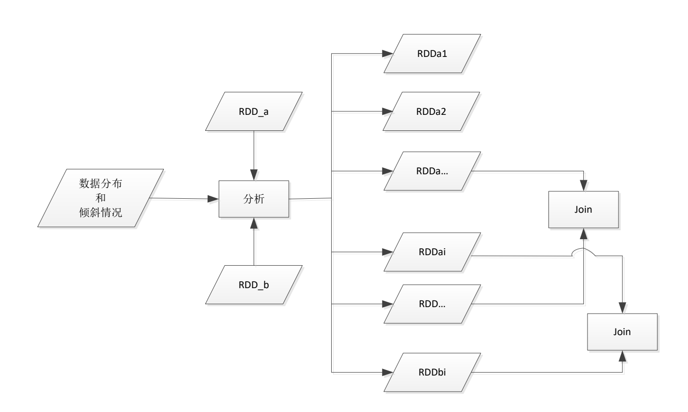


图 3　算法第三阶段-分段预连接并组装连接结果

表 1　图中所用符号说明

符号	描述
RDD_a（b）	RDD a(数据表 a，数据表 b)
JoinKey_a(b)	RDD a(b)的连接属性
$BF_{A(B)}$	RDD a(b)的 BloomFilter 位数组
BF	BloomFilter 位数组
RDD_a(b)_f	分区后的 RDD_a(b)
$RDD_{ai(bi)}$	RDD_a(b)拆分出的分区
JoinKey_a(b)_f	对 RDD_a(b)_f 抽取 Joinkey 操作获得的集合

1.2 采样分析-统计数据分布

对过滤后的两个分区 RDD_a_f 和 RDD_b_f 分别抽取 Joinkey 操作获得 JoinKey_a_f 和 JoinKey_b_f 两个集合。对 JoinKey_a_f 和 JoinKey_b_f 两个 RDD 使用 sample 算子，分别对两个分区的 Joinkey 进行采样（如果采样率太大会造成数据量很大的计算负担，根据帕累托法则，将采样率定为 0.2），得出两份样本，然后对两个样本进行统计分析量，计算出来两张表中数据量最大的是哪几个 key 和相应的数据倾斜情况。

1.2.1 采样分析

设两个表中较小的表为 RDD_S，下面对 RDD_S 连接属性 Key 进行采样分析。

给定采样率 β，对 RDD_S 进行采样得到样本集合 SampleKeysRs 并分析 SampleKeysRs 样本中 Key 的倾斜情况，依据 Key 频次分布和 Key 整体分布进行划分为两类互不相交的集合，把频次分布频次高的放在集合 S_{stew}，频次相对均匀或者频次低的放在集合 S_{dis} 中。RDD_S 中出现未在 SampleKeyRs 包含的 Key，依据统计概率，属于频次很低的 key，也放入 S_{dis} 中。

$$RDD_S_{Key} = S_{stew} \cup S_{dis} \quad (1)$$

采样操作使用 Spark 提供的 ReservoirSample(水塘采样算法)进行操作,其目的在于从包含 n 个项目的集合 S 中选取 k 个样本，其中 n 为一很大或未知的数量，尤其适用于不能把所有 n 个项目都存放到主内存的情况，使用 ReservoirSample 可以减少采样占用的内存空间。

1.2.2 统计数据分布

对于采样得到的两个样本集 JoinKey_a_s 和 JoinKey_b_s 分别进行统计分析和对比，主要目标是查看两个 RDD 中 key 对应数据分布情况和数据倾斜情况。由于 Spark Statistics 库中有概括统计、分层统计、核密度估算等方法，故使用 Spark Statistics 进行统计分析。对于单个样本集进行统计分析主要是为了查看单个 RDD 中数据的倾斜情况，和估算单个表中 Key 对应数据总量。对于两表的样本集统计分析结果进行对比是为了找出两个分区中相同 Key 值的数据分布情况。在做统计分析时，利用 Spark Statistics 库中的方法，得到如下统计信息：a)单个 RDD 单个 Key 占的比例 R_k；b)单个 RDD 数据倾斜严重的 Key；c)单个 RDD 中 Key 按照采样样本中的量的自然排序；d)两个 RDD 中相同 Key 的数据比例 Rs。

1.3 分段预连接并组装连接结果

对两表中较小的表 RDD_S, 进行 Key 抽样，得到 Key 抽样样本集合 SampleKeysRs。根据 SampleKeyRs 中 Key 的倾斜情况和 RDD_B_UK 中 Key 的分区号对 RDD_S 进行拆分。拆分过程使用贪心算法把 RDD_S 动态拆分成一定数目的子集 RDD_S_i。每次拆分出一个子集 RDD_S_i 就进行 RDD_B 与该子集的连接，连接后再拆分并连接，直至所有 RDD_S 数据连接完成。这一过程中都是每次拆分出 RDD_S_i 后再连接再拆，因而每次的 RDD_S_i 的数据量相对于 RDD_S 很小，无须占用大量内存空间和计算量。下面分别对单侧表拆分方法、预连接和连接结果的组装进行了详细的描述。

1.3.1 单侧表拆分方法

对预连接得到的 PreJoined，需要根据 PreJoined 中 Key 包含的分区号进行重划分，因此 PreJoined 中包含的分区号的均匀分布和整体性是影响划分并行度的因素。而每个分区号对应的所有 Key 的数据倾斜情况是直接影响连接组装各个计算节点负载量，为此选取的贪心策略主要参考以下三点：

a)拆分的子集 RDD_S_i 的所有 Key 包含的分区号要均匀分布，每个 Key 对应的分区号数量上要基本相等。所有 Key 对应的分区号整体性要接近 RDD_B 的所有分区号整体性。

b)任意一个分区上对应的所有 Key 的数据规模要和其他分区上接近。

c)对于上述两点，贪心策略主要依据 RDD_B_UK 中 Key 所对应分区列表及 RDD_S 的 Key 抽样分析划分的集合 S_{stew} 和 S_{dis} 作启发选择。

整个拆分过程是使用贪心算法把 RDD_S 动态拆分成一定数目 n 的子集 RDD_S_i，$0 \le i < n$。拆分过程是 n-1 次运算，每次使用贪心策略每次对输入的 RDD_S_j 拆分出其 RDD_S_i，其中，$0 \le j < n$。

数据表 RDD_S_j 和 RDD_S_i 满足以下关系：

始化输入 RDD_S_j 的值为 RDD_S

$$RDD_S = RDD_S_j \cup RDD_S_i$$

$$RDD_S_j = RDD_S_{j+1} \cup RDD_S_i$$

$$i, j = n-1, RDD_i = RDD_j$$

拆出 RDD_S 的子集 RDD_S_i 后，就用 RDD_B_UK 和 RDD_S_i 进行连接阶段，连接阶段包括预连接和组装连接结果。重复上述过程直至 RDD_S 中所有 Key 都被拆分出来，并进行过连接。

1.3.2 预连接

RDD_B_UK 和 RDD_S 的子集 RDD_S_i 进行并行连接得到连接结果 PreJoined，PreJoined 中每个元素包含了一个 Key 和 RDD_S_i 中对应于该 Key 的元组，以及该 Key 对应的 RDD_B 分区号的列表。经过拆分的 RDD_S_i 的数据规模远小于 RDD_S，且 RDD_B_UK 的数据量也很小，故预连接的速度很快，并且预连接的结果分布在计算结果中，减少了 Shuffle 的开销。

1.3.3 组装连接结果

将 PreJoined 数据进行按照 Key 所对应的分区号的重划分，而且分区号与 RDD_B（选定较大的表 RDD_B，并且生成连接属性 Key 与分区号列表的无重集，就是为了减少连接过程中数据重划分的量）分区号是一一对应，划分得到的结果集和 RDD_B 存储在相同的分区，之后再通过 zipPartitions 操作将

RDD_B 和 PreJoined 做快速的组装。这一过程没有在网络上传输 RDD_B 中的数据。且前面选择的贪心策略会使这一过程中数据倾斜度降低，各个节点负载基本平衡，且分区号的分布均匀，这种整体性能够充分利用集群节点的并行计算能力，提高计算速度。

2 算法分析

2.1 网络 I/O 代价分析

本连接算法中，单表去重阶段没有网络开销，网络 I/O 开销来自广播 BloomFilter 进行过滤和连接阶段的预连接过程的数据划分和连接结果的组装，如式(2)所示：

$$NetCost = NetCost_{bf} + NetCost_{pj} + NetCost_{zp} \quad (2)$$

算法虽然是拆分单侧表，但所有拆分并连接的网络 I/O 数据量等于单侧表网络 I/O 数据量，且每次拆分并连接的网络 I/O 数据量远小于单侧表网络 I/O 数据量造成的网络压力。其中 $NetCost_{pj}$ 是预连接数据划分的网络 I/O 代价，用数据量表示，如式(3)所示：

$$NetCost_{pj} = Size(RDD_B_UK) + \alpha \bullet Size(RDD_S) \quad (3)$$

由于过滤去重后的 RDD_B_UK 中元组个数少于 RDD_B，且每个元组仅包含一个 Key 值及其对应的分区号，因此 RDD_B_UK 的数据量远小于 RDD_B，即：

$$Size(RDD_B_UK) = \alpha \bullet \varepsilon \bullet Size(RDD_B)$$

其中，$0 < \alpha \ll 1, 0 < \varepsilon \ll 1$

$NetCost_{zp}$ 是结果组装的网络 I/O 代价，通常由于预连接之后相当于 RDD_B_UK 在连接属性上又做了全局的去重且排除了不能连接的元组，$NetCost_{zp}$ 会远低于 $NetCost_{pj}$。而 $NetCost_{bf}$ 是使用 BloomFilter 进行两表过滤生成的压缩位数组的网络 I/O 代价，因此数据量也远小于 $NetCost_{pj}$。

设参与连接的节点数为N，则总的通信量将由这些节点分摊，因此本文算法的网络 I/O 代价估算结果如式(4)所示：

$$Net\,Cost \approx Net\,Cost_{pj} + Net\,Cost_{zp} + Net\,Cost_{bf} \le \frac{\alpha \bullet \varepsilon \bullet Size(RDD_B) + Size(RDD_S)}{N} \quad (4)$$

其中，$0 < \alpha \ll 1, 0 < \varepsilon \ll 1$

2.2 内存空间代价分析

算法中构建的 BloomFilter 数据结构在过滤完表后就释放，不需要缓存在内存中。整个拆分过程中需要缓存过滤后两表 Key 及其分区号和 Key 的倾斜集合 S_{stew}，并且预连接的结果也需要缓存，故占用了一定的内存消耗。S_{stew} 是根据给定抽样率对过滤后的 RDD_S 的倾斜数据的 Key 的集合，因此 S_{stew} 内存销号远小于 Size(RDD_S),所以算法的内存代价如式(5)所示：

$$\begin{aligned} MemCost &= 2 \times Size(RDD_B_UK) + \alpha \bullet Size(RDD_S) + Size(S_{stew}) \\ &\approx 2 \times Size(RDD_B_UK) + \alpha \bullet Size(RDD_S) \end{aligned} \quad (5)$$

2.3 对比分析

由于 Spark Broadcast Join 只能应用于两表中有一表是小表的场景，因此本文只和目前应用较为广泛的 Hash Join 方式作对比分析。Hash Join 是重划分连接，所以每个节点的网络通信量为 (Size(RDD_B)+Size(RDD_S))/N， 故所需内存空间为 Size(RDD_B)+Size(RDD_S)。本文算法与 Spark Hash Join 算法的对比如表 2 所示。

表 2 两种等值连接算法代价对比分析

等值链接算法	网络 I/O 代价	内存空间代价
Hash 连接	$\frac{Size(RDD_B) + Size(RDD_S)}{N}$	$Size(RDD_B) + Size(RDD_S)$
本文算法	$\le \frac{\alpha \bullet \varepsilon \bullet Size(RDD_B)}{N} + \frac{Size(RDD_S)}{N}$	$\approx \alpha \bullet Size(RDD_S) + 2 \times Size(RDD_B_UK)$

3 实验分析

实验选用了 2 组数据表，每组都是 2 个数据大小不同的数据表（第一组数据倾斜程度低于第二组），给定的抽样率为 40%分别使用本文算法和 Spark 的 Hash Join 算法对数据进行了关联操作，并作对比分析。两表的关联字段分别是 C_RID 和 P_RID。数据表大小如表 3 所示：

表 3 数据集大小

数据集	1	2
表 1	7.2G	7G
表 2	8.6G	9.1G

3.1 实验环境

实验环境使用的软件名称，如表 4 所示。

表 4 软件环境

操作系统	JDK 版本	Hadoop	Spark
CentOS7-x64	Oracle jdk1.8.0_92	Apache Hadoop 2.7	Apache Spark 2.1.0

本文的硬件环境是在实验室的计算机上使用 VMWare 创建的 CentOS7 虚拟集群环境中进行的，集群中共有 6 个节点，每个虚拟节点硬件配置如表 5 所示。

表 5 硬件环境

CPU	内存	硬盘	网络
2 核 2.6 GHz	2 GB	60 GB	100 Mbps

3.2 实验结果

两组实验的运行结果如图 4 所示。

图 4 中的执行时间是 3 次执行取得的平均值，单位为 min。通过实验结果本文连接算法的执行速度仍然比 Spark SQL 中的

Hash Join 高出 1 倍以上，并且通过图 4 中两组实验中使用同一种算法的执行速度对比，当两组实验的数据表规模变化不大时，如果数据倾斜情况不一样，使用 Spark Hash Join 得到的两次时间变化比较明显，尤其是第二组实验的数据集倾斜情况比第一组数据严重，对应的 Hash Join 执行时间明显比第一组满了 4.8 min；而本文算法的执行速度变化不大，本文算法第二组实验比第一组仅慢了 0.7 min。通过以上分析，可见本文的算法在做连接操作时，不仅比 Spark Hash Join 操作的执行速度快，而且在面对数据倾斜情况依然具有变化不大的执行效率。

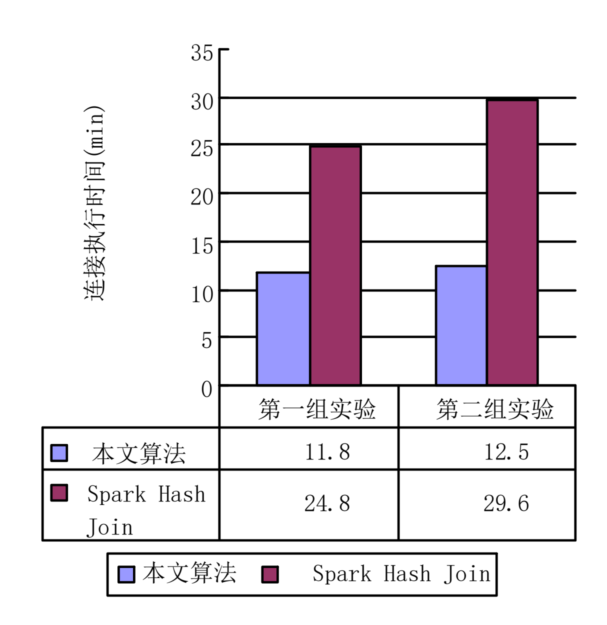


图 4　连接执行时间对比

4　结束语

目前 Spark 在数据分析中已经普及，交互式大数据分析业务需求越来越多。链接性能一直是交互式数据分析中的性能瓶颈。本文通过对目前 Spark/MapReduce 上的等值连接算法进行了分析研究，提出了一种基于 Spark 的等值连接过程优化算法。本算法有很好的适用性，从大数据价值密度低特征和数据倾斜情况出发，过滤、去重、拆分连接过程，并通过实验和分析证明了比现有的连接算法的性能提升，且对数据倾斜具有适应性。

参考文献：

[1] Spark [EB/OL]. http://spark. apache. org/.

[2] 于俊，向海，代其锋，等. Spark 核心与高级应用 [M]. 北京：机械工业出版社, 2016.

[3] Armbrust M, Xin R S, Lian C, et al. Spark SQL: relational data processing in Spark [C]// Proc of ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2015: 1383-1394.

[4] Blanas S, Patel J M, Ercegovac V, et al. A comparison of join algorithms for log processing in MaPreduce [C]// Proc of ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2010: 975-986.

[5] Xu Y, Zhou X, Chen L, et al. Handling data skew in parallel joins in shared-nothing systems [C]// Proc of ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2008: 1043-1052.

[6] Ramesh S, Papapetrou O, Siberski W. Optimizing distributed joins with bloom filters [C]// Proc of International Conference Distributed Computing and Internet Technology. 2008: 145-156.

[7] Zhou J, Wang Q, Gao J, et al. An approach for load balancing in MapReduce via dynamic partitioning [J]. Journal of Computer Research & Development, 2013.

[8] Gufler B, Augsten N, Reiser A, et al. Load balancing in mapreduce based on scalable cardinality estimates [C]// Proc of IEEE International Conference on Data Engineering. Washington DC: IEEE Computer Society, 2012: 522-533.

[9] Gufler B, Augsten N, Reiser A, et al. Handling data skew in MapReduce [C]// Proc of International Conference on Cloud Computing and Services Science. 2011: 574-583.

[10] 卞昊穹，陈跃国，杜小勇，等. Spark 上的等值连接优化 [J]. 华东师范大学学报：自然科学版, 2014, 2014 (5): 263-270.

[11] Hacigumus H, Iyer B, Mehrotra S. Providing database as a service [C]// Proc of International Conference on Data Engineerin. 2002: 29-38.

[12] 王卓，陈群，李战怀，等. 基于增量式分区策略的 MapReduce 数据均衡方法 [J]. 计算机学报, 2016, (1): 19-35.

[13] 聂长海，蒋静. 覆盖表生成的可配置贪心算法优化 [J]. 软件学报, 2013 (7): 1469-1483.